UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Tecnura

http://revistas.udistrital.edu.co/ojs/index.php/Tecnura/issue/view/650

DOI: http://dx.doi.org/10.14483/udistrital.jour.tecnura.2015.3.a07

INVESTIGACIÓN

Técnica para solución de recurrencias, usada en el análisis de la complejidad de algoritmos recursivos

Technique for recurrences solving, used in recursive algorithm complexity analysis

Luis Emilio Montenegro Salcedo[1], Luz Deicy Alvarado Nieto[2]

Fecha de recepción: 13 de septiembre de 2014 **Fecha de aceptación:** 18 de agosto de 2015

Cómo citar: Montenegro Salcedo, L. E., & Alvarado Nieto, L. D. (2015). Técnica para solución de recurrencias, usada en el análisis de la complejidad de algoritmos recursivos. *Revista Tecnura*, 19, 89-97. doi: 10.14483/udistrital.jour.tecnura.2015.SE1.a07

Resumen

Este artículo presenta un método alternativo, directo y poco común para solucionar recurrencias de primer orden, tanto homogéneas como no homogéneas; aplicable a ecuaciones que representan el comportamiento de algoritmos recursivos. Dicho método se asocia al funcionamiento computacional del algoritmo, facilitando su comprensión y el análisis de la complejidad. El proceso se ilustra con ejemplos de ecuaciones correspondientes a algoritmos muy conocidos y frecuentemente utilizados.

Palabras Clave: Algoritmos, análisis de algoritmos, cantidad de operaciones, complejidad algorítmica, eficiencia computacional, recurrencias, recursividad.

Abstract

This paper shows an alternative, direct and unusual method to solve both first order recurrences homogeneous and inhomogeneous; applicable to equations representing the behavior of recursive algorithms. Such method is associated with the performance of the computational algorithm, by facilitating the understanding and complexity analysis. The process is illustrated by examples of equations for well-known and frequently used algorithms.

Keywords: Algorithms, algorithmic complexity, Analysis of Algorithms, computational efficiency, operations number, recurrences, recursion.

[1] Licenciado en educación especialidad matemáticas, magister en auditoría de Sistemas y computación, especialista en edumática. Docente de la Universidad Francisco José de Caldas. Bogotá, Colombia. Contacto: emontenegro@udistrital.edu.co

[2] Ingeniera de Sistemas, magister en Ingeniería de Sistemas, Doctor en Informática, con énfasis en Ciencias de la Computación e Inteligencia Artificial. Docente de la Universidad Francisco José de Caldas. Bogotá, Colombia. Contacto: lalvarado@udistrital.edu.co

INTRODUCCIÓN

Desde el punto de vista computacional, cuando se busca la solución de un problema es usual utilizar algoritmos, los cuales debido a los avances tecnológicos se espera que sean correctos, sencillos, claros, eficaces, eficientes y oportunos (Baase, 1999, 2002).

El análisis algorítmico busca determinar el comportamiento de los algoritmos en términos de memoria y tiempo (Aho, 1983, Brassard, 1997). Para ello, se define una medida del tamaño del problema (la cual depende de la naturaleza de dicho problema) -llámese N- y se busca una función f(N) que refleje el comportamiento del algoritmo con base en el tamaño definido (McConnell, 2007, Sedgewick, 2013). Es importante aclarar que el análisis se hace para un N muy grande. Cuando se define en términos de tiempo, la función es comúnmente denominada t(N).

Ejemplos de funciones que representan el comportamiento de algoritmos hay muchos, entre ellos se pueden mencionar: $t(N) = N + log\ N$, $t(N) = N + 5$, $t(N) = N^2 + 3N + 3$, $t(N) = a^N + N + 10$.

Estas funciones, que evidencian de manera general el consumo de recursos por parte del algoritmo a medida que N crece, se pueden agrupar en "familias" representativas, de tal manera que es posible afirmar, por ejemplo, que un algoritmo tiene comportamiento exponencial, logarítmico, lineal, cuadrático, cúbico, etc. (Aho, Hopcroft, 1983, Baase, 1999, 2002, Brassard, 1997, Sedgewick, 2013).

Particularmente, en el caso de los algoritmos recursivos el comportamiento se expresa mediante relaciones lineales de recurrencia con coeficientes constantes (Baase, 1999, 2002, McConnell, 2007, Sedgewick, 2013). Una relación lineal de recurrencia implica que t(N) se expresa de la siguiente forma: $t(N) = F(N) + a_1 t(N-1) + a_2 t(N-2) + \cdots a_c t(N-c)$.

De manera particular, si en una relación lineal de recurrencia se encuentra que $F(N) = 0$, se dice que esta relación es homogénea. De lo contrario, dicha relación se considera no homogénea (Aho, 1983, Brassard, 1997).

En la definición de t(N) como una función de recurrencia, fácilmente se puede observar que la misma función t aparece aplicada a uno o varios valores de N, como ocurre en los siguientes casos: $t(N) = t(N-1) + 1$, $t(N) = 2\ t(N-1) + N$.

Los ejemplos anteriores corresponden a relaciones lineales de recurrencia de primer orden, pues el valor actual $t(N)$ depende solamente del valor inmediatamente anterior $t(N-1)$. Sin embargo, algunos algoritmos también presentan comportamientos que pueden ser representados con relaciones lineales de recurrencia de segundo orden, en las cuales aparece dos veces la función t, es decir, el valor actual $t(N)$ depende de los dos valores anteriores $t(N-1)$ y $t(N-2)$ (Sedgewick, 2013). Un caso concreto de esto es la representación del comportamiento para el algoritmo recursivo que halla el término n-ésimo de la sucesión de Fibonacci: $t(N) = t(N-1) - t(N-2)$.

Como se evidencia fácilmente, las relaciones de recurrencia están expresadas en términos recursivos, es decir, en términos de la misma función, pero con uno o varios tamaños menores. Sin embargo, en el análisis de algoritmos se espera que la función esté planteada únicamente en términos del tamaño del problema, es decir, hallar una expresión no recursiva de t(N). Por tal motivo se hace indispensable aquí solucionar de la manera más sencilla y clara las relaciones de recurrencia, de forma que las ecuaciones sean expresadas en términos de N.

La primera parte de este artículo muestra algunas funciones de recurrencia para algoritmos típicos, posteriormente presenta ejemplos de la aplicación de técnicas tradicionales en la solución de relaciones de recurrencia tanto homogéneas como no homogéneas y, finalmente, propone un método inductivo, directo y concreto para hallar la solución de las recurrencias antes mencionadas.

FUNCIONES QUE REPRESENTAN EL COMPORTAMIENTO DE ALGORITMOS RECURSIVOS COMUNES

En su libro Algoritmos en C++ (Sedgewick, 1995), el autor analiza el tema "recurrencias básicas" y

presenta lo que él denomina "fórmulas", que describen los comportamientos más comunes de algunos algoritmos recursivos; entre ellas se encuentran las siguientes: $t(n) = t(n-1) + n$ para $n \geq 2$ y $t(1) = 0$. En este caso se describe el comportamiento de un algoritmo que efectúa una operación sobre todos los datos de entrada para descartar uno de ellos, luego hace una llamada recursiva disminuyendo dicho dato. Un claro ejemplo de esto es el algoritmo de burbuja recursivo, en donde usando un ciclo se ubica el mayor o el menor elemento del arreglo, luego se hace una llamada recursiva disminuyendo en 1 el tamaño de dicho arreglo, es decir, tomando un dato menos.

$t(n) = t(n/2) + 1 \ \ para \ n \geq 2 \ \ con \ \ t(1) = 1.$ La función de recurrencia aquí presentada describe un algoritmo que hace una operación fundamental y posteriormente hace una llamada recursiva para la mitad de los datos de entrada, es decir, se queda solamente con la mitad de los datos. Un ejemplo de algoritmos que tienen este comportamiento es el de búsqueda binaria o la búsqueda en un árbol binario ordenado y balanceado.

$t(n) = t(n-1) + 1 \ \ para \ n \geq 2 \ \ con \ \ t(0) = 0.$ Esta función de recurrencia representa el comportamiento de un algoritmo que realiza una única operación, la cual permite descartar un dato. Posteriormente hace una llamada recursiva, descartando dicho dato. Un ejemplo de este comportamiento lo presenta el algoritmo que efectúa el cálculo del factorial de manera recursiva.

$t(n) = 2t(n/2) + n \ \ para \ n \geq 2 \ \ con \ t(1) = 0.$ E n este caso, la recurrencia describe un algoritmo que hace dos llamadas recursivas, cada una con la mitad de los datos y antes, en medio o después de dichas llamadas, hace un recorrido por todos los datos. Un claro representante de este comportamiento es el algoritmo Mergesort; también podría considerarse el Quick sort, pero solo en el mejor caso, cuando el pivote siempre se ubica en la mitad de los datos.

$t(n) = 2t(n/2) + 1$ para $n \geq 2$ con $t(1) = 0$. Finalmente, esta recurrencia describe un algoritmo que realiza una operación, divide los datos en 2 y hace una llamada recursiva para cada una de las partes de la división. Los algoritmos recursivos de recorrido de un árbol binario balanceado o el conteo de sus nodos son algoritmos que ejemplifican este comportamiento. También puede considerarse dentro de este comportamiento el peor caso de búsqueda en un árbol binario no ordenado.

TÉCNICAS TRADICIONALES DE SOLUCIÓN DE RECURRENCIAS

En general, se han establecido y generalizado dos métodos para resolver recurrencias, dependiendo de si estas clasifican como homogéneas o no (Aho, 1983, Brassard, Bratley, 1997, Cormen, 2009, Sedgewick, 2013).

Recurrencias homogéneas

Para este caso se debe aplicar el siguiente procedimiento:

- Intuir o predecir cuál es el tipo de función que puede describir el comportamiento de la recurrencia. Pasa por una sospecha acerca de la complejidad posible del algoritmo. Por ejemplo, se asume que tiene un componente de tipo exponencial en las recurrencias de segundo orden.
- Se sustituye la función predicción en la recurrencia obteniendo la ecuación característica, la cual corresponde a las raíces de un polinomio.
- La solución de la ecuación característica permite encontrar un conjunto de soluciones particulares, de las cuales se puede inferir una familia de soluciones posibles.
- Se aplican las condiciones iniciales (correspondientes a las salidas no recursivas del algoritmo), con el fin de seleccionar el ejemplar más apropiado de esa familia, obteniéndose así la solución de la recurrencia.

El ejemplo típico de recurrencias homogéneas presentado en diferentes libros (Aho, Hopcroft, 1983, Brassard, 1997, Cormen, 2009, Sedgewick, 2013), es el algoritmo recursivo que permite

encontrar el término n-ésimo de la sucesión de Fibonacci, partiendo de las semillas 0 y 1, ecuación (1).

$$t(n) = \begin{cases} t(n-2) + t(n-1) & Si \;\; n > 1 \\ n & Si \;\; n = 0 \;\; o \;\; n = 1 \end{cases} \qquad (1)$$

- Se intuye, especialmente al ver el comportamiento del algoritmo, que la función debe ser de orden exponencial. Por tanto, la predicción es: $t(n) = a^n$ siendo $a > 0$.
- Se observa que la recurrencia de la ecuación (1) es homogénea, dado que se puede escribir como $t(n) - t(n-2) - t(n-1) = 0$ para el caso general, *n > 1*.
- Al sustituir $t(n) = a^n$ en la ecuación homogénea se convierte en:

$$a^n - a^{n-2} - a^{n-1} = 0 \qquad (2)$$

- Que equivale a: $a^{n-2}(a^2 - a - 1) = 0$. Al aplicar la condición $a > 0$ se reduce a la llamada ecuación característica $a^2 - a - 1 = 0$.
- Las soluciones particulares se muestran en las ecuaciones (3) y (4).

$$t_1(n) = \left(\frac{1-\sqrt{5}}{2}\right)^n \quad y \qquad (3)$$

$$t_2(n) = \left(\frac{1+\sqrt{5}}{2}\right)^n \qquad (4)$$

- A partir de las cuales se expresa la familia de soluciones mediante la ecuación (5).

$$t(n) = C_1\left(\frac{1-\sqrt{5}}{2}\right)^n + C_2\left(\frac{1+\sqrt{5}}{2}\right)^n \qquad (5)$$

- Como t(n) debe satisfacer las condiciones iníciales *t(0) = 0* y *t(1) = 1,* al sustituir n en la ecuación (5) se concluye que la solución es como se muestra en la ecuación (6).

$$t(n) = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n \qquad (6)$$

El método anterior se puede aplicar a todas las recurrencias homogéneas y se convierte en un proceso meramente algebraico, aplicable sin mayores inconvenientes.

Recurrencias no homogéneas

Sin embargo, para el caso de recurrencias no homogéneas, que entre otras cosas son las que más se presentan en los algoritmos recursivos, es necesario convertirlas a homogéneas aplicándoles un tratamiento matemático, el cual varía dependiendo de la forma que tenga dicha recurrencia. Para este caso se consideran los siguientes ejemplos típicos:

Recurrencia del factorial

El algoritmo para calcular recursivamente el factorial de un número entero mayor o igual a cero da lugar a la recurrencia: $t(n) = t(n-1) + 1 \quad Si \;\; n > 0 \;\; con \;\; t(n) = 0 \;\; cuando \;\; n = 0$. La recurrencia se escribe en la ecuación (7).

$$(n) - t(n-1) = 1 \qquad (7)$$

Al no ser homogénea, con el término independiente constante, la ecuación (7) se puede desarrollar de la siguiente forma:

- Se remplaza *n* por *n-1*, convirtiéndose en la ecuación (8).

$$t(n-1) - t(n-2) = 1 \qquad (8)$$

- La diferencia de las ecuaciones (7) y (8) se puede escribir $t(n) - 2t(n-1) + t(n-2) = 0$

Teniendo así una recurrencia homogénea con ecuación característica $x^2 - 2x + 1 = 0$.

Como la solución es múltiple *x = 1*, la solución general se expresa en la ecuación (9).

$$t(n) = c_1 1^n + n c_2 1^n \qquad (9)$$

O simplemente:

$$(n) = c_1 + c_2\, n \qquad (10)$$

Como solo se tiene una condición inicial y se requieren dos para encontrar las constantes, se puede aplicar la recurrencia y hallar *t(1) = t(0) + 1 = 1*.

Por tanto, $c_1 = 0 \;\; y \;\; c_2 = 1$ y al sustituir en la ecuación (10) se encuentra la solución mostrada mediante la ecuación (11).

$$t(n) = n \tag{11}$$

Recurrencia del Quick sort

Para el algoritmo Mergesort y el mejor caso del algoritmo Quick sort se deduce la recurrencia: $2t(n/2) + n \;\; Si \;\; n > 1, con \;\; t(0) = 1$

Con el fin de convertir la recurrencia $t(n) = 2t(n/2) + n$ a homogénea, es necesario hacer dos cosas:

- La primera es un cambio de variable, $n = 2^m$ en cuyo caso se restringe el proceso a los valores de n que sean potencias de 2. Entonces la recurrencia se escribe como se muestra en la ecuación (12).

$$t(2^m) = 2t(2^{m-1}) + 2^m \tag{12}$$

La cual también se puede escribir en forma simplificada:

$$t_m - 2t_{m-1} = 2^m \tag{13}$$

Haciéndose evidente que no es homogénea.

- La segunda acción es hacer que desaparezca el término 2^m en la ecuación (13). De modo similar al ejemplo anterior, en este caso se remplaza m por *m-1* y luego se multiplica por 2 (ecuación (14)).

$$2t_{m-1} - 4t_{m-2} = 2^m \tag{14}$$

- Al realizar la diferencia entre las ecuaciones (13) y (14).

$$t_m - 4t_{m-1} + 4t_{m-2} = 0 \tag{15}$$

La recurrencia (15) tiene ecuación característica $x^2 - 4x + 4 = 0$ y. por tanto, $t_1 = 2^m \;\; y \;\; t_2 = m2^m$. Con solución general $t_m = c_1 2^m + c_2 m 2^m$

Al retornar a la variable inicial $n = 2^m$ o su equivalente $m = log_2 n$

$$t(n) = c_1 n + c_2 n log_2 n \tag{16}$$

Aplicando la condición inicial en la ecuación (16), se concluye que $c_1 = 0$

La recurrencia original 3 se puede escribir:

$$n = t(n) - 2t(n/2) \tag{17}$$

Como (16) es una solución de la ecuación (17) se sustituye obteniendo la ecuación (18).

$$n = (c_1 n + c_2 n log_2 n) - 2(c_1 \frac{n}{2} + c_2 \frac{n}{2} log_2 \frac{n}{2} \tag{18}$$

Cuya simplificación es $n = c_2 n$, para concluir que c_2 debe ser igual a 1. Entonces la ecuación (19) muestra la solución correspondiente a las constantes $c_1 \; y \; c_2$:

$$t(n) = n log_2 n \tag{19}$$

Por supuesto, con el desarrollo de este apartado no se pretende aportar nada nuevo. Solo resaltar algunos ejemplos que ilustran la gran variedad de alternativas que pueden darse al solucionar recurrencias no homogéneas y lo artificiosos que pueden ser estos métodos.

PROPUESTA

A continuación, se presenta un proceso algebraico general para solucionar recurrencias no homogéneas, el cual está íntimamente ligado al comportamiento del correspondiente algoritmo. No solo es una forma más sencilla de solución, sino que además facilita la comprensión del funcionamiento del correspondiente algoritmo recursivo.

Recurrencia del factorial

En este caso se retoma la fórmula $t(n) = t(n-1) + 1 \quad Si \quad n > 0 \quad y \quad t(n) = 0 \quad cuando\, n = 0.$

Dado que para cualquier n > 0

$$(n) = t(n-1) + 1 \tag{20}$$

Mientras que el parámetro de la función sea mayor que cero, se pude aplicar la fórmula de recurrencia, obteniéndose, a partir de la ecuación (20), la siguiente secuencia de remplazos:

Remplazo 1: $t(n) = t(n-1) + 1$ Llamado recursivo para calcular (n-1)!

Remplazo 2:

$t(n) = [t(n-2) + 1] + 1 = t(n-2) + 2$ Llamado para calcular *(n-2)!*

Remplazo 3:

$t(n) = [t(n-3) + 1] + 2 = t(n-3) + 3$ Llamado para calcular (n-3)!

Remplazo 4:

$t(n) = [t(n-4) + 1] + 3 = t(n-4) + 4$ Llamado para calcular (n-4)!

⋮

Remplazo k: $t(n) = [t(n-k) + 1] + (k-1)$) Llamado para calcular (n-k)!

En general, se tiene:

$$(n) = t(n-k) + k \tag{21}$$

El anterior conjunto de ecuaciones resulta al efectuar la sencilla sustitución de cada nuevo valor en la ecuación original. Además, cada línea describe el siguiente llamado recursivo. De tal forma que la ecuación (21) permite interpretar que en el paso k-ésimo, el algoritmo ha hecho k multiplicaciones y queda pendiente el cálculo de la cantidad de multiplicaciones restantes dado por t(n-k), necesarias para calcular el factorial de n-k.

Se sabe que el algoritmo terminará los llamados y tendrá una salida no recursiva, en el paso k (caso base para algoritmos recursivos) correspondiente al llamado para hallar 0!, es decir, cuando se cumpla que:

$$t(n-k) = t(0) \tag{22}$$

En tal caso se tendrá el número total de multiplicaciones requeridas para calcular el factorial del número entero positivo n dado.

La ecuación (22) se logra cuando k es igual a n, y se podrá reescribir la ecuación (21) de la siguiente forma: $t(n) = t(0) + n$ Luego la solución a la recurrencia es la que aparece en la ecuación (23).

$$t(n) = n \tag{23}$$

Recurrencia del Quick sort

$$t(n) = \begin{cases} 0 & Si \quad n = 1 \\ 2t(n/2) + n & Si \quad n > 1 \end{cases} \tag{24}$$

Para resolver esta recurrencia se procede en la misma forma. Para cada término $t\left(\frac{n}{2}\right)$ se aplica la fórmula de recurrencia con la condición de que $\frac{n}{2}$ sea diferente de 1 (n > 1), obteniéndose:

Llamado 1: $t(n) = 2t\left(\frac{n}{2}\right) + n$

Describe las n comparaciones requeridas para clasificar el arreglo en los menores o mayores del pivote, más las comparaciones necesarias para ordenar los dos subarreglos resultantes, de tamaño $\frac{n}{2}$ cada uno.

Llamado 2: $t(n) = 2\left[2t\left(\frac{n}{4}\right) + \frac{n}{2}\right] + n \;\; = 4t\left(\frac{n}{4}\right) + 2n$

2n es el acumulado de comparaciones para clasificar el arreglo inicial y los dos subarreglos de $\frac{n}{2}$ cada uno y $4t\left(\frac{n}{4}\right)$ corresponderá a la cantidad de comparaciones requeridas para ordenar 4 subarreglos de tamaño $\frac{n}{4}$ cada uno.

Llamado 3: $t(n) = 4\left[2t\left(\frac{n}{8}\right) + \frac{n}{4}\right] + 2n \;\; = \;\; 8t\left(\frac{n}{8}\right) + 3n$

3n es la cantidad de comparaciones para clasificar el arreglo inicial de tamaño n, los dos subarreglos de tamaño $\frac{n}{2}$ y los cuatro de $\frac{n}{4}$. Ahora quedan por ordenar 8 subarreglos de tamaño cada uno.

Se generaliza la fórmula para el k-ésimo llamado recursivo.

$$t(n) = 2^k t\left(\frac{n}{2^k}\right) + kn \tag{25}$$

La expansión máxima de la recursividad se logra en el paso k-ésimo, para el cual el arreglo se reduce a un elemento, $\frac{n}{2^k} = 1$ (ya no hay nada que ordenar). En ese momento la salida es no recursiva y se habrán realizado todas las comparaciones requeridas.

$\frac{n}{2^k} = 1$, equivale a $2^k = n$; por tanto, se deduce que $k = log_2 n$ y la ecuación (25) se puede escribir de la siguiente forma: $t(n) = nt(1) + nlog_2 n$. Por último, como t(1) = 0, la recurrencia tendrá la solución presentada en la ecuación (26).

$$t(n) = nlog_2 n \qquad (26)$$

Otros ejemplos

A continuación se presentan otros ejemplos de recurrencias para ilustrar la solución por este método.

Ordenamiento por el método de burbuja recursivo

$$t(n) = \begin{cases} 0 & Si \;\; n = 1 \\ t(n-1) + (n-1) & Si \;\; n > 1 \end{cases} \qquad (27)$$

Si el tamaño del arreglo es n = 1, se requieren 0 comparaciones, t(1) = 0, ya está ordenado. Para tamaños de n mayores que 1, se realizan n-1 comparaciones para encontrar el mayor (o el menor) y mediante un llamado recursivo se ordenan los n-1 elementos restantes, realizándose t(n-1) comparaciones.

Aplicando el método propuesto a la recurrencia de la ecuación (27), se tiene:

Remplazo 1:
$t(n) = t(n-1) + (n-1)$
Remplazo 2:
$t(n) = [t(n-2) + (n-2)] + n - 1 = t(n-2) + (n-2) + (n-1)$
Remplazo 3:
$t(n) = [t(n-3) + (n-3)] + (n-2) + (n-1)$
Equivale a:
$t(n) = t(n-3) + (n-3) + (n-2) + (n-1)$
⋮

Generalizando el proceso:

Remplazo k:
$t(n) = t(n-k) + (n-k) + \cdots + (n-3) + (n-2) + (n-1)$

El proceso termina en el k-ésimo llamado, para el cual el tamaño del arreglo se reduce a un elemento, es decir, cuando n-k = 1, o simplemente k = n-1. En tal caso la recurrencia se reescribe:

$$t(n) = t(1) + 1 + 2 + 3 + \cdots + (n-3) + (n-2) + (n-1) \qquad (28)$$

Es decir, la ecuación (28) se resume en $t(n) = \sum i,$ y por tanto, se tiene como solución la ecuación (29).

$$t(n) = \frac{n(n-1)}{2} \qquad (29)$$

Búsqueda en un árbol binario ordenado y balanceado

Este comportamiento también puede corresponder al algoritmo de búsqueda binaria en un arreglo, siendo t(n) la cantidad máxima de comparaciones requeridas.

$$t(n) = \begin{cases} 1 & Si \;\; n = 1 \\ t(n/2) + 1 & Si \;\; n > 1 \end{cases} \qquad (30)$$

Si el árbol o el arreglo tienen solo un elemento, se requiere una comparación, t(1) = 1. Para una cantidad de elementos n mayor que 1, $t(n) = t\left(\frac{n}{2}\right) + 1$, es decir, una comparación con el elemento de la raíz (o el elemento medio) y de no coincidir, el proceso de búsqueda se reduce a la mitad de los elementos. Los menores o los mayores, pero no ambos. En este caso al aplicar a la ecuación (30) el proceso de solución planteado se obtiene:

Remplazo 1:
$t(n) = t(n/2) + 1$, corresponde al primer llamado recursivo.
Remplazo 2:
$t(n) = [t(n/4) + 1] + 1 = t(n/4) + 2$, segundo llamado recursivo.
Remplazo 3:
$t(n) = [t(n/8) + 1] + 2 = t(n/8) + 3$, tercer llamado recursivo.

⋮

Remplazo k: $t(n) = t\left(\frac{n}{2^k}\right) + k$.

En este caso, el proceso termina cuando $\frac{n}{2^k} = 1$, es decir, $k = log_2 n$ y la recurrencia se expresa $t(n) = t(1) + log_2 n$. Por tanto, la solución es la ecuación (31).

$$t(n) = 1 + log_2 n \tag{31}$$

Búsqueda en un árbol binario balanceado no ordenado

Para este caso se puede plantear la recurrencia en la siguiente forma:

$$t(n) = \begin{cases} 1 & Si \;\; n = 1 \\ 2t(n/2) + 1 & Si \;\; n > 1 \end{cases} \tag{32}$$

La diferencia con el anterior (apartado 4.3.2), consiste en que al no estar ordenado el árbol, puede implicar hacer la búsqueda sobre los dos subárboles, cada uno con aproximadamente $n/2$ elementos y, por tanto, la recurrencia planteada corresponde al peor caso de la búsqueda, cuando el elemento no está o por casualidad corresponde al último elemento posible de revisar. El desarrollo de la ecuación (32) será:

Remplazo 1:
$t(n) = 2t\left(\frac{n}{2}\right) + 1$

Remplazo 2:
$t(n) = 2\left[2t\left(\frac{n}{4}\right) + 1\right] + 1 = 4t\left(\frac{n}{4}\right) + 2 + 1$

Remplazo 3:
$t(n) = 4\left[2t\left(\frac{n}{8}\right) + 1\right] + 2 + 1 = 8t\left(\frac{n}{8}\right) + 4 + 2 + 1$

⋮

En general, para k:
$t(n) = 2^k t\left(\frac{n}{2^k}\right) + 2^{k-1} + \cdots + 4 + 2 + 1$

La recurrencia se puede escribir $t(n) = 2^k t\left(\frac{n}{2^k}\right) + 2^k - 1$. Dado que $\sum_{i=1} 2^i = 2^k - 1$. Cuando $\frac{n}{2^k} = 1$ o $2^k = n$, se logra $t(n) = nt(1) + n - 1$

Obteniéndose la solución en la ecuación (33):

$$t(n) = 2n - 1 \tag{33}$$

RESUMEN DEL PROCESO

El método para resolver recurrencias que se justifica y presenta en este artículo se puede resumir de la siguiente forma:

- Se escribe la fórmula de recurrencia por resolver, especialmente si es no homogénea, que es el más difícil de solucionar, como se muestra en el apartado 3.2. Para el caso de recurrencias homogéneas, la solución es más mecánica, dado que siempre se sigue el mismo proceso algebraico.
- Se remplaza paso a paso en forma reiterada, aprovechando que la función es la misma.
- Se analiza su comportamiento y se generaliza, para cualquier paso K. Este punto es el más importante y debe llevar a simplificar por alguna propiedad matemática conocida.
- Por último, se aplica la condición inicial, la cual equivale a la salida no recursiva; esto permite encontrar la solución de la recurrencia, como una función en términos únicamente de n.

CONCLUSIONES

Los algoritmos recursivos se utilizan cada vez con mayor frecuencia al ser soluciones elegantes, simples, modulares y bien estructuradas a problemas complejos. Para analizar el comportamiento de dichos algoritmos es necesario solucionar las ecuaciones de recurrencia que lo representan.

Como puede observarse, las recurrencias son indispensables en los procesos de análisis de algoritmos. Los métodos tradicionalmente usados para su solución tienden a desestimular a algunos ingenieros estudiantes, debido a la gran variedad de casos y formas de solución que se pueden presentar.

El método aquí expuesto plantea seguir el mismo proceso de solución en todos los casos y

generalmente no se requieren conocimientos muy avanzados de matemáticas. Dicho método promueve la observación y refuerza la exigencia de generalizar comportamientos. Siendo esta una destreza que se presume inherente a los diseñadores de algoritmos.

En resumen, a lo largo de este documento se ha presentado y justificado un proceso cuyo desarrollo resulta más natural para los ingenieros, debido a que está especialmente ligado a la lógica del algoritmo al cual corresponde la recurrencia.

REFERENCIAS

Aho, A., Hopcroft, J., Ullman, J. (1983). *Data structures and algorithms*. Massachusetts, U.S.A.: Addison Wesley.

Baase S., Van, A. (1999). *Computer Algorithms: Introduction to design and analysis*. Massachusetts, U.S.A: Addison Wesley.

Baase, S., Van, A. (2002). *Algoritmos computacionales: Introducción al análisis y diseño*. México DF, México: Addison Wesley.

Brassard, G., Bratley, P. (1997). Fundamentos de algoritmia. Madrid, España: Prentice Hall.

Cormen, T., Leiserson, C., Rivest, R., Stein, C. (2009). *Introduction to algorithms*. Massachusetts: Massachusetts Institute of Technology.

McConnell, J. (2007). *Analysis of Algorithms: an active learning approach*. Ontario, Canada: Jones and Bartlett Publisher.

Sedgewick, R. (1995). *Algoritmos en C++*. Massachusetts, E.U.A.: Addison-Wesley/Díaz de Santos.

Sedgewick, R., Flajolet, P. (2013). *An introduction to the analysis of Algorithms*. Massachusetts: Pearson Education.